A Toolbox for Isophase-Curvature Guided Computation of Metrology Holograms

User Manual

Ulf Griesmann, April 27, 2020

Copyright and Disclaimers

The full description of the software and procedures described in this User Manual requires the identification of certain commercial products and their suppliers. The inclusion of such information does not indicate that these products or suppliers are endorsed by the National Institute of Standards and Technology (NIST), or are recommended by NIST, or that they are necessarily the best materials or suppliers for the purposes described.

1. Introduction

This user manual describes the installation and use of a toolbox for the creation of computer-generated holograms (CGHs) for interferometric precision surface metrology (in the following referred to as the “CGH toolbox”) [1]. The manual describes the core toolbox functions that must be called to calculate the layout of a hologram that can be submitted to a fabrication facility. Finally, several extensively commented examples illustrate the functionality of the toolbox and the ways in which the toolbox can be adapted for specific measurement needs. Reference [1] describes the details of the algorithms that were developed for the CGH toolbox.

2. Installation Procedures and Prerequisites

In addition to the CGH toolbox itself, users must install two open-source toolboxes, one that provides functions for manipulating planar polygons (“polygon-toolbox”), and a second one that provides functions for the creation of lithographic layout files in GDSII format (“gdsii-toolbox”). The installation procedures for the required components are described below.

2.1 Gnu Octave

The CGH toolbox requires an installation of the GNU Octave software [2]. The toolbox was tested with Octave versions 4.4 and 5.2. In addition to the basic installation of Octave, the “optim” package, which provides functions to solve optimization problems, and the “parallel” package, which provides functions for parallel processing on multi-core computers, must be installed following the instructions on the Octave web site. The “parallel” Octave package currently does not work on Windows versions of Octave. The CGH toolbox will still work without the parallel package, but it will not be possible to take advantage of multi-processing.

2.2 CGH toolbox

All files of the CGH toolbox are stored in a compressed archive with the name `cgh-toolbox-<version>.zip`. The first step of the installation process is to unpack the archive into a convenient directory, or folder, e.g.:

```
~/Toolboxes
```

Then, in Linux or MacOS, change to the toolbox directory

```
cd ~/Toolboxes/cgh-toolbox
```

and, at a shell prompt, run the

```
./makemex-octave
```

script in the top level directory of the toolbox.. This will create two “Matlab extension” (.mex) files that are needed for some of the toolbox functions. In Windows, the .mex functions of the CGH toolbox must be compiled from within Octave. Start Octave and change the current working directory to the top level directory of the CGH toolbox. Then run

```
>>makemex
```

to create the necessary .mex files.

2.3 GDSII toolbox

Holograms calculated by the CGH toolbox can be converted into layout files in the Graphics Design System Information Interchange (GDSII) format used by most electron-beam or laser lithography systems using an open-source GDSII toolbox for Octave and Matlab [3]. GDSII layout files can be used to fabricate holograms in research foundries, or they can be submitted to commercial fabrication services. The installation process for the GDSII toolbox is the same as for the CGH toolbox. The archive of the GDSII toolbox (.zip file) is unpacked and the `makemex-octave` script in the top level directory of the toolbox is executed at a shell prompt in Linux or MacOS. In Windows, the `makemex` script in the toolbox top level directory must be run from within Octave.

2.4 Polygon toolbox

The polygon toolbox primarily consists of external (.mex) functions that must be compiled before they can be used by Octave. The process is the same as for the CGH toolbox.

2.5 Setting the Octave search path

All toolboxes must be contained in the Octave search path. This is accomplished by including the line

```
addpath( genpath(‘FULL NAME OF TOOLBOX DIRECTORY’) );
```

to the Octave startup file `.octaverc` for each of the toolboxes. An example of a startup file with the required modifications is shown in the appendix. Once the changes to the startup file are made, Octave must be restarted. The correct installation of the toolboxes can be verified by issuing the following commands at the Octave prompt:

```
>>cgh_version
>>gdsii_version
>>polygon_version
```

In each case the toolbox version is displayed in the format

```
Interpreter version   : Octave 4.4
GDSII Toolbox version : 144  (2019-August 18)
```

if the toolbox installation was successful. An error message at this stage is most likely the result of an incorrect entry in one of the search paths in the startup file, which prevents the toolboxes from being found. The correct inclusion of the toolboxes in the search path can also be verified with the `path` command in Octave.

2.6 Layout viewers

Layout files in GDSII format can be evaluated with many commercial software tools that are available for editing and viewing electronics design layouts. Klayout is an open-source tool that works well for the diffractive optics layouts created by the CGH toolbox [4].

3. Core Functions

From a user of toolbox, the creation of a metrology hologram with the CGH toolbox requires three function calls to create a domain tiling (`polytile`), to define options and parameters for the CGH (`cghparset`), and, finally, to generate the CGH (`phase2cgh`). These three function calls are typically combined in a script file that is specific to each application of the CGH toolbox. Several example scripts are distributed with this user manual.

3.1 polytile

The `polytile` function generates a rectangular tiling that covers the hologram domain to enable, among other things, the efficient calculation of large holograms on multi-core computers (see discussion in Ref. [1]). The use of this function is illustrated in the `demo_tiling.m` script in the directory `Examples/tiling`.

`[stile] = polytile(bnd_out, bnd_inn, tilpar, plotpar=[]);`	
Input	
`bnd_out`	The two-dimensional domain of a hologram has an outer boundary described by a polygon of arbitrary length. The polygon is specified as a m×2 matrix where every row contains the coordinates of a polygon vertex.
`bnd_inn`	The hologram domain can have inner boundaries, which are also specified as m×2 matrices. When the domain has more than one inner boundary, the argument `bnd_inn` must be a cell array of polygons. Inner boundaries may overlap, and an inner boundary may overlap with the outer boundary.
`tilpar`	This argument specifies the size and positioning of the tiles in a tiling. In the simplest case, `tilpar` is a scalar that specifies the width of square tiles. Rectangular tiles can be generated by assigning `tilpar` a 1×2 vector containing the width and the height of the tiles. `tilpar` can also be a structure with an offset field that makes it possible to control the placement of the tiling relative to the coordinate origin (see help text of `polytile`).
`plotpar`	For diagnostic purposes the tiling can be plotted, with or without tile numbers. The values in the `plotpar` structure control various aspects of the tiling plot.
Output	
`stile`	A tiling is described as a structure array where each tile is specified by the coordinates of its lower left corner (`stile(k).llc`) and its upper right corner (`stile(k).urc`).

3.2 cghparset

Parameters and options for the CGH computation are combined in a structure. The `setcghpar` function is a convenient way to set the parameters and tolerances and to provide default values.

```
cghpar = cghparset('par1',val1, 'par2',val2, …);
cghpar = cghparset(oldpar, 'par1',val1, 'par2',val2, …);
```

Input

The arguments to this function must be specified as a variable number of keyword-value pairs. An existing parameter structure is modified when it is passed as the first argument. Keyword-value pairs can be specified in any order. Unspecified parameters are assigned a default value.

Keyword	**Value**	
`algorithm`	A two-character string that selects the algorithm used to calculate the hologram. 'pa' selects the pilot approximation algorithm, 'fi' the isophase following algorithm, and 'hi' the algorithm for phase functions with Hilbert phase terms. See Ref. [1] for a discussion of the algorithms. **The default value is 'fi'**.	
`vphase`	A 1×2 vector containing the phase boundaries of "opaque" Fresnel half-zones. Specifying $[\phi_1,\phi_2]$ implies all phase boundaries $[\phi_1,\phi_2]+2\pi k, k\in\mathbb{Z}$. The areas of the hologram domain where the phase function is within the specified phase boundaries are returned as closed polygons. **The default value is [0,π]**.	
`tol`	A structure containing the tolerances that control the computation of holograms. Default values are assigned to unspecified fields. **Default values assume μm length unit.** The structure has the following fields:	
	`vertex`	The position tolerance of the polygon vertices. **The default is 10^{-4} length units**. The length unit will typically be the μm, and the corresponding default tolerance is 0.1 nm, sufficient for even the most advanced e-beam lithography systems.
	`segdev`	Phase boundary curves are approximated by polygons that deviate from the curve between vertices. Polygon vertices are spaced along the boundary curve such that the deviation is equal to `segdev`. **The default value is 5×10^{-3} length units** (5 nm when length unit is μm).

	`seglen`	The boundary detection algorithm may fail near points where the curvature of a phase boundary changes sign. In this case the polygon segment length must be given an upper limit, e.g. 10 μm. **The segment length is not limited by default**.
	`maxit`	The precise location of vertices on phase boundary polygons is found using an iterative level-finding algorithm. This field sets an upper limit for the number of iterations. **The default limit is 150 iterations**. Far fewer iterations are typically required to find polygon vertices.
	`maxbs`	Before the position of a vertex on a phase boundary curve can be calculated, two points on either side of the boundary must be found such that they bracket the vertex point along a line normal to the phase boundary. The field value limits the number of bracket search iterations. **The default value is 32**.
	`hderiv`	Numerical derivatives of phase functions are calculated using a symmetric secant approximation. This field specifies the spacing of the points at which the phase function is evaluated to compute derivatives. **The default is 15×10^{-3} length units**. Very small values can result in numerical instability.
	`discon`	For phase functions with discontinuities, this field specifies the tolerance with which the discontinuity is located. **The default is 15×10^{-3} length units**.
	`maxpol`	The maximal number of polygon segments with identical phase that are connected by the phase filling algorithm. **The default value is 10**. It is rarely necessary to join more than two or three segments. A large number of segments usually indicates an error.
`grid`	A structure defining the phase function sampling grids for tile areas and tile edges that are needed to calculate initial approximations for isophase lines and their edge intersections. The structure has the following fields:	
	`area`	A 1×2 vector with number of samples in the x- and y-directions. **The default value is [100,100]**.

	`edge`	Number of samples along tile edges. **The default value is 10000**.
`sing(k)`	A structure array with information regarding phase singularities in phase functions with Hilbert terms. All singularities must be located on tile edges. Phase functions can have multiple singularities, but each tile can have only one singularity on one of its edges. The default value is an empty matrix, []. If present, the structure (array) must have the following fields:	
	`pos`	A 1×2 vector with the coordinates of the singularity in the plane.
	`era`	A scalar with an exclusion radius around the singularity that is used in detecting the proximity of the singularity to a phase boundary curve. A value of 0.05 (50 nm when the design unit is μm) works well. **No default value**.
`aperture(k)`	A hologram calculated on a tiling will generally extend beyond the desired boundaries of the hologram. This field contains a structure array with polygons and and Boolean set operations that can be applied to the polygons describing the output hologram. The structures have the following fields.	
	`poly`	A closed polygon (m×2 matrix of vertices) for a Boolean set operation.
	oper	`aperture(k).oper` is a string specifying a Boolean set operation that will be applied to the set of hologram polygons (first operand) and the polygon defined in `aperture(k).poly` (second operand). Available set operations are 'or' (set union), 'and' (set intersection), 'notb' (set difference), and 'xor' (exclusive difference). When more than one operation is defined, the order in which they are applied matters. **No default value**.
`debug`	This variable is set to `false` by default, and any errors that occur during the calculation of a hologram layout in a tile area are trapped and the tile number is displayed. When the value is set to `true` Octave displays the error location in the usual way. **Default value is `false`**.	
Output		
`cghpar`	A structure containing all tolerances and parameters.	

3.3 phase2cgh

This function is called to carry out the computation of a hologram.

`cap = phase2cgh(stile,fphase,phapar,cghpar,ncpu=[],verbose=0);`	
Input	
`stile`	A structure array with a rectangular tiling of the hologram domain; usually the output of the `polytile` function.
`fphase`	A function handle of the phase function. Phase functions must have the form `fphase(X,Y,phapar)`, where `X` and `Y` are vectors with coordinates in the hologram plane, and `phapar` is a structure with phase function parameters. The CGH toolbox provides a set of commonly used phase functions, e.g. for polynomial models that can be used as templates for new, problem specific phase functions.
`phapar`	A structure with phase function parameters.
`cghpar`	A structure with tolerances and parameters for a CGH; usually the output of the `cghparset` function.
`ncpu`	An optional argument to specify the number of processors, or logical processing units, that are used for the CGH computation. All available processors are used by default (as long as the Octave parallel package is available).
`verbose`	A optional argument that controls the detail of progress information that is displayed. The default value is 0 (no information). When the argument is set to 1, a message is displayed every time the processing of a new tile begins. A value of 2 results in much more detailed information that is primarily useful for debugging
Output	
`cap`	A cell array of closed polygons circumscribing hologram fringes.

4. Debugging Strategies

Locating errors (“bugs”) in software that is executed with a high degree of concurrency on multi-processor computers can be a challenge. When the calculation of the hologram for a specific tile area fails it can be difficult to identify the tile on which the error occurred. The CGH toolbox can assist with the location of errors. By default any errors that occur during the calculation of the hologram for an individual tile are trapped and an error message together with the number of the tile on which the error occurred is displayed. The tile number information can be used to restrict the computation to the tile that causes the problem. For example, if an error occurs on tile number 157, the following line can be added after the call to the `polytile` function which generates a tiling:

```
tiles = polytile(…);
tiles = tiles(157);
```

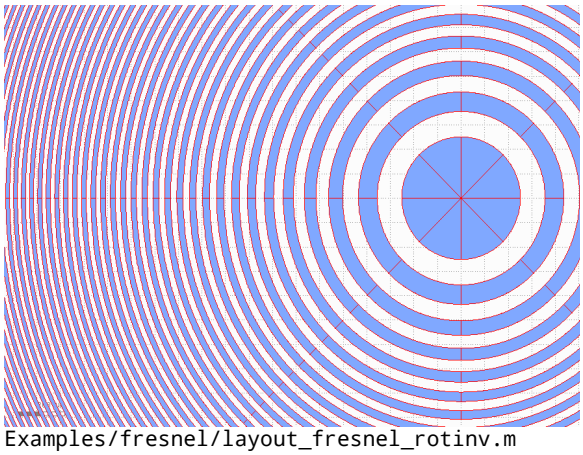
When the script is executed again, the hologram is calculated only for tile 157. In addition, the debug flag in the CGH parameters should be set to `true.` Errors will then no longer be trapped, and a function call trace that precisely locates the error and its cause will be displayed instead.

5. Examples

This CGH toolbox user manual is accompanied by a set of example scripts which demonstrate a range of toolbox applications. In this section we provide brief discussions of the example scripts together with illustrations of their output, which can be used to ascertain the correct function of the CGH toolbox.

5.1 Fresnel Zone Lenses

The directory `Examples/fresnel` contains scripts that generate a Fresnel zone lens. The zone lens has a diameter of 20 mm and creates a focus, in first diffraction order, at 100 mm distance from the zone plate when it is illuminated with collimated light from a helium-neon laser. Two of the scripts, with suffixes ‘pa’ and ‘fi’ illustrate the use of different algorithms for the computation of the

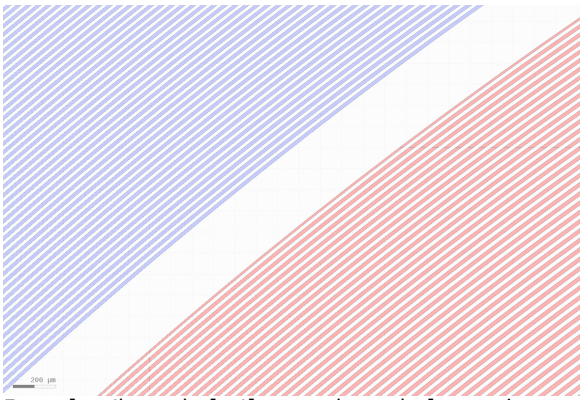


Examples/fresnel/layout_fresnel_rotinv.m

zone lens. The zone lens hologram can also be computed much faster by taking advantage of the rotational symmetry of the zone lens. This alternative approach is illustrated in the script with the 'rotinv' suffix. Comparing the hologram layouts generated by these scripts verifies that Fresnel lens holograms calculated with the algorithms provided by the CGH toolbox are identical to holograms based on the Fresnel zone radii that can be found in basic optics texts [5].

5.2 Rotationally Invariant Holograms

Form errors of rotationally invariant (aspheric) surfaces can be measured with rotationally invariant holograms (although this is not necessarily the best choice). When a radial phase function model is available, the hologram layout can be calculated in the same simplified way as the Fresnel zone lens in Sec. 5.1. The example script `layout_hyperbola_rotinv.m` in the directory `Examples/hyperbola` provides an illustration. The script calculates two rotationally invariant holograms, a ring-shaped alignment hologram that is used to align the CGH in the converging test beam of a Fizeau interferometer, and a hologram for generating the test wavefront, in this case for a concave hyperbolic mirror. The radial phase functions for both holograms are polynomial models that were calculated with the commercial ray-tracing software used to model the measurement setup. For illustration purposes the holograms are calculated with two different methods. The alignment hologram is calculated with the simplified approach suitable for rotationally invariant phase functions, while the main hologram is calculated with the isophase following algorithm of the CGH toolbox. A phase function for a rotationally symmetric test part can also be modeled as a two-dimensional polynomial. This was done in the script `layout_asphere_doe.m` in `Examples/asphere`.



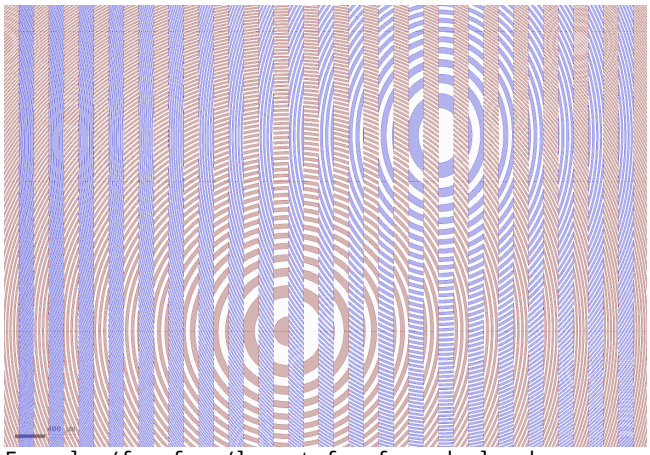
`Examples/hyperbola/layout_hyperbola_rotinv.m`

5.3 Holograms with Tilt or Decenter

Holograms used for surface metrology applications are often not rotationally symmetric, because it is more difficult to block stray light that is created by the higher diffraction orders of a binary hologram when the hologram is symmetric about the optical axis. An increased amount of stray light tends to degrade the measurement result, leads to "hot spots" at the center of the interferometer's field-of-view, and increases the measurement uncertainty. Separation of diffraction orders is easier to achieve when the hologram is either tilted or de-centered by a small amount in the test

beam of the interferometer, which is equivalent to adding a tilt carrier frequency to the hologram. The range of line widths in a hologram with tilt carrier is much smaller than in a hologram with a point of stationary phase, which makes the hologram easier to fabricate.

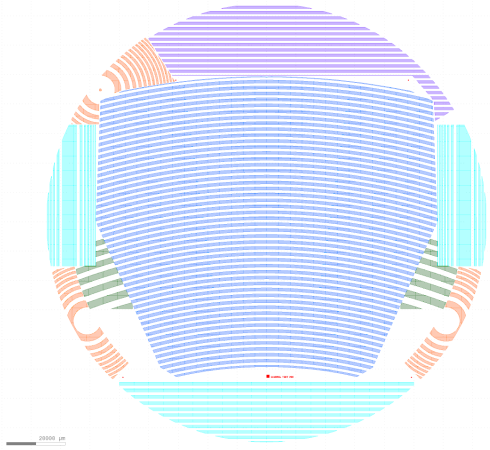
5.4 Multiplexed Holograms



Examples/freeform/layout_freeform_dual_cgh.m

For measurements that require a low measurement uncertainty it may be desirable to encode two wavefronts into a hologram, one spherical wavefront that can be used to calibrate the interferometer errors, and the non-spherical test wavefront. The simplest way to encode more than one wavefront into a hologram is through spatial multiplexing in which different areas of the hologram contain sections of different holograms. An example of this approach is given in the Octave script `layout_dual_freeform_cgh.m` in directory `Examples/freeform`. In this example, two holograms are interleaved in 100 µm wide strips. One hologram generates a spherical wavefront, and the second one generates the test wavefront for a freeform surface. Both wavefronts have an angular offset so that both wavefronts can be aligned individually to a reference sphere or the test part with the freeform surface.

5.5 Compound Holograms

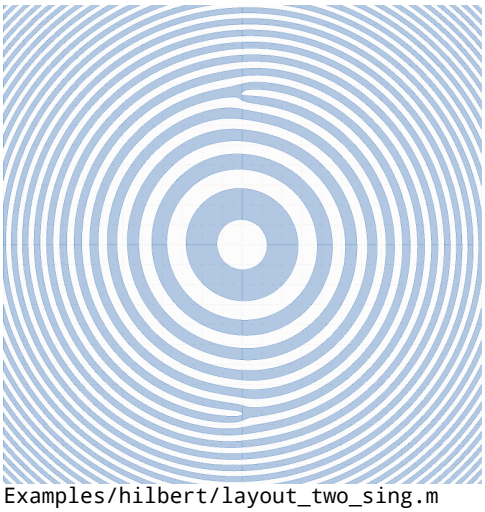


Examples/multiple/Generate_cgh.m

A metrology hologram for surface form metrology will consist of several sub-holograms with different functions in almost all applications. For example, in addition to the hologram that creates a test wavefront, one or more holograms may be required to align the hologram substrate in the test beam of an interferometer. A simple example is the hologram created with the `layout_hyperbola_rotinv.m` script (see Sec. 5.2), which has a ring-shaped alignment hologram to position and align the substrate in the beam of a Fizeau objective (transmission sphere). A more

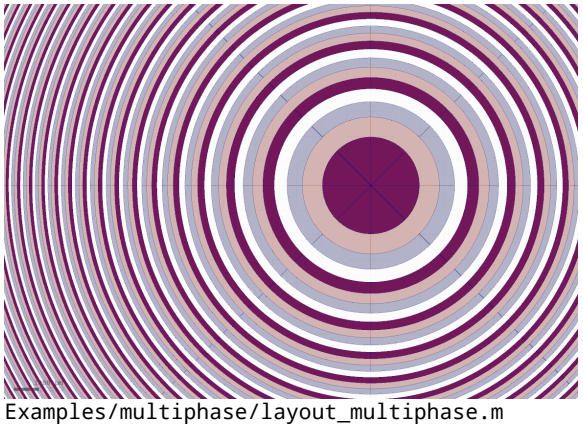
elaborate example is in `Examples/multiple`. This hologram contains 9 sub-holograms needed for alignment purposes. Several of the holograms have a large line density and their computation is time consuming. A “divide-and-conquer” strategy for handling challenging compound holograms is illustrated in the scripts `Generate_cgh.m` and `Assemble_cgh.m`. In the design of a complex test setup, the phase function of the holograms is often revised multiple times. The script shows how the computation of holograms can be made more efficient by, for example, restricting the computation to a subset of holograms that have been revised. The example script also shows how an artificially large wavelength can be used to better visualize the Fresnel zones of a hologram, or to rapidly compute a “draft version” of a hologram, which is helpful during the design stage of a test setup.

5.6 Holograms with Hilbert Terms


Examples/hilbert/layout_two_sing.m

Phase functions can have phase singularities and discontinuities (branch cuts), of which Hilbert phase terms are a prominent example. Hilbert phase functions describe optical beams with helical wavefronts that are finding an increasing number of applications in optics [1]. A Hilbert phase term, when combined with a Fresnel phase, results in characteristic bifurcations of Fresnel zones, and the concentric zones of a Fresnel zone lens are transformed into spirals. The CGH toolbox can create hologram layouts from phase functions containing one or more Hilbert terms, with some limitations. A tile of the CGH domain tiling can only have one singularity, which must be located on the tile edge, and the tiling must be designed to meet this requirement. A second requirement is that that the discontinuity must not coincide with a tile edge.

5.7 Multi-Level Holograms


Examples/multiphase/layout_multiphase.m

Holograms used for surface form metrology are generally binary holograms because they have the lowest fabrication cost. Fabrication of phase holograms with higher diffraction efficiency requires lithography processes with multiple, overlayed exposures. The script in `Examples/multiphase` illustrates how the CGH toolbox may be used to create layouts for

holograms with multiple phase levels. Generally, N exposures, followed by an etching process, can create holograms with 2^N phase levels. The script `layout_multiphase.m` illustrates this for a hologram with four phase levels. The layouts for the two exposure steps are placed on two different GDSII layers. The etch depth following the first exposure must be twice the etch depth following the second exposure. The phase function used in the example script `layout_multiphase.m` is the same function that was used for the main hologram in `layout_hyperbola_rotinv.m`.

5.8 Runtimes

The time required for the computation of a hologram depends on a multitude of variables. Hologram area and line density are among the most important, as is the degree of parallel execution that can be achieved on the available computing platform. The following table contains the execution times for some of the example scripts to help users estimate the computing resources needed for their own applications. The third column refers to the number of Octave instances running in parallel to calculate sections of the CGH on different tiles, which was chosen to be the same as the number of available logical central processing units (CPUs). For similar processors, the runtime scales roughly in inverse proportion to the number of instances.

Script	Platform / Architecture	Instances	Runtime
`asphere/ layout_asphere_doe.m`	Intel i7-3960X / x86_64	12	60 m 37 s
	Intel Xeon E5-2699 / x86_64	72	11 m 37 s
`freeform/ layout_freeform_dual_cgh.m`	Intel i7-3960X / x86_64	12	20 m 17 s
	Intel Xeon E5-2699 / x86_64	72	4 m 18 s
	Amlogic S922X / ARM64 v8	6	1 h 15 m
`synchrotron/ layout_sn1_cgh.m`	Intel Xeon E5-2699 / x86_64	72	29 s
	Intel i7-3960X / x86_64	12	2 m 8 s
	Amlogic S922X / ARM64 v8	6	6 m 16 s
`multiple/ Generate_cgh.m`	Intel Xeon E5-2699 / x86_64	72	93 m 43 s

Appendix: A sample Octave initialization file .octaverc

This is an example of a user-specific initialization, or startup, file for Octave. This file is a text file that can be created with any text file editor, including the file editor built into Octave. The initialization file has the name ".octaverc" on Linux, MacOS, and Cygwin and is located in a user's home directory. For the native Windows version of Octave, the startup file is typically located in the directory "C:\Users\USERNAME".

```
%--- beginning of startup file ----------------------------------
warning("off", "Octave:possible-matlab-short-circuit-operator");

% set toolbox paths
addpath( genpath('/home/ulfg/Toolboxes/cgh-toolbox') );
addpath( genpath('/home/ulfg/Toolboxes/polygon-toolbox') );
addpath( genpath('/home/ulfg/Toolboxes/gdsii-toolbox') );

% load always needed packages (optional)
pkg load optim
pkg load parallel % currently not on Windows
pkg load symbolic

% default font sizes (optional)
set(0, 'defaultaxesfontname', 'Noto Sans');
set(0, 'defaultaxesfontsize', 14);
set(0, 'defaulttextfontname', 'Noto Sans');
set(0, 'defaulttextfontsize', 16);

% better looking prompt (optional)
PS1('octave:\#> ');
PS2('> ');
PS4('+ ');

% turn off pager (optional)
more off
%--- end of startup file ----------------------------------
```

Note that in Windows the search paths must be preceded by a drive letter, e.g.

```
   addpath( genpath('c:/home/ulfg/Toolboxes/cgh-toolbox') );
```

or

```
   addpath( genpath('c:\home\ulfg\Toolboxes\cgh-toolbox') );
```

References

1. Griesmann U, Soons JA, and Khan GS (2019) *A Toolbox for Isophase-Curvature Guided Computation of Metrology Holograms,* J. Res. NIST
2. Eaton JW *et al.* (1996-2019) A high-level interactive language for numerical computations. Available at https://octave.org
3. Griesman U (2008-2019) An Octave and Matlab toolbox for layout files in GDSII format. Available at https://github.com/ulfgri/gdsii-toolbox
4. Köfferlein M (2006-2019) Klayout – High performance layout viewer and editor. Available at https://www.klayout.de
5. Hecht E (2016) *Optics, 5th Edition*